International Journal of Scientific Engineering and Technology
Volume No.5 Issue No.10, pp: 477-481

ISSN:2277-1581
01 October 2016

Use of Enhanced Transparent Data Encryption to Protect Database Against Exposure of Backup Data

Mr. Nizigiyimana Alain; Dr. Ann KIBE; Dr Cheruiyot W.K.
Jomo Kenyatta University of Agriculture and Technology -Kigali Campus
Corresponding Email: nizigiyimanapuma@yahoo.fr

***Abstract**: This study aimed at using Transparent Data Encryption uses to protect database backups against exposure of backup data. The study a more improved data base security was expected through encrypting database backups. Results from this study revealed that The TDE algorithm for database backup protection was designed and implemented improved database backup security. This was echoed by the penetration test performed which showed that databases backups without data TDE have higher vulnerability risks than databases with TDE. Therefore, protecting backup data with TDE can prevent the exposure of data via backups*.

Keywords:TDE, database backup, database exposure, database protection

I. Introduction

In organisations database Administrators or other IT in charge of databases in company often do the daily backup of databases by copying physical files and put somewhere in computer location which is online/offline. In that case it is exposed to packet sniffing over network. This exposed data is at high risk of being attacked. Therefore, there is an urgent need of protection of these backup data. To ensure maximum security of organization's databases. The present study is proposed in order to attempt this problem by developing an enhanced transparent data encryption which will be a powerful tool in protecting database backups.

Present day life is vastly driven by Information Technology. Extensive use of IT is playing vital role in decision-making in all commercial and non-commercial organizations. All activities are centred on data, its safe storage and manipulation. In present scenario organizations' survival is at stake if its data is misused. Data is vulnerable to a wide range of threats like, Weak authentication, Backup Data Exposure, Denial of Service, etc. (Jimmy Thakkar,2015)

Transparent Data Encryption (TDE) shields database up to considerable extent against such threats. TDE is used to prevent unauthorized access to confidential database backups, reduce the cost of managing users and facilitate privacy managements. This latest technology arms users' i.e. database administrators to solve the possible threats to security of data. This technology allows encrypting databases on hard disk and on any backup media. TDE nowadays, is the best possible choice for bulk encryption to meet regulatory compliance or corporate data security standards (Khaleel et al., 2011).

Encryption is said to occur when data is passed through a series of mathematical operations that generate an alternate form of that data; the sequence of these operations is called an algorithm. To help distinguish between the two forms of data, the unencrypted data is referred to as the plaintext and the encrypted data as cipher text. Encryption is used to ensure that information is hidden from anyone for whom it is not intended, even those who can see the encrypted data. The process of reverting cipher text to its original plain text is called decryption. (Khaleel et al., 2011).

As noted above, TDE's specific purpose is to protect data at rest by encrypting the physical files of the database, rather than the data. These physical files include the database file (.mdf), the transaction log file (.ldf) and the backup files (.bak).

The protection of the database files is accomplished through an encryption key hierarchy that exists externally from the database in which TDE has been enabled. The exception to this is the database encryption key, which was introduced to the database encryption key hierarchy specifically to support the TDE feature, and is used to perform the encryption of the database files.

The key hierarchy, and their required location of each key, is illustrated. The service master key exists at the instance level. The database master key and certificate at the Master database are used to protect the database encryption key that is located at the user database, which is the "*Home Lending"* database in our example. The database encryption key is then used to decrypt the database files of the user database.

The Transparent Data Protection is viewed as the most effective components to protect database against exposure of backup data. Some Database Administrators or other IT in charge of databases in company can do the daily backup of databases by copying physical files and put somewhere in computer location which is online/offline. In that case it is exposed to packet sniffing over network.

Though it is advisable to protect backup through encryption, it has been observed that most companies do not protect their backups. While Transparent Data Encryption (TDE) shields database up to considerable extent against any kind of threat (Basharat et al., 2012).

Therefore, this research mainly focuses on the use of enhanced Transparent Data Encryption to protect databases against exposure of backup data instead of protecting the original database only used in production.

ISSN:2277-1581
01 October 2016

After realizing that companies do not protect their backup data which are exposed attack, the proposed research study is aimed at using of enhanced Transparent Data encryption to protect back up data. The Success of this research will benefit Organizations and other institutions to protect their backups and hence increasing the security of their most valuable data set such as: Corporate data, Customer data, Financial data.

In addition, this study will also provide an awareness of necessity of backup data protection for IT administrators and managers.

The rest of the paper illustrates the algorithm, results of the penetration test and conclusion

II. Material and Methodology

Application of Transparent Data Encryption algorithm to backup data protection

As noted previously, TDE prevents the backup files from being opened by a plain text editor. It also limits the recovery of the database backup file to the instance that holds the encryption key hierarchy that was in existence at the time the backup was created.

Backup files of databases with TDE enabled are encrypted using a key hierarchy that includes the service master key of the SQL Server instance, the database master key and certificate for the Masterdatabase.

Despite this dependency, none of these keys are included with the standard database backup, and must be backed up separately via the following commands:

- BACKUP SERVICE MASTER KEY to backup of the service master key.
- BACKUP MASTER KEY to backup of a database master key.
- BACKUP CERTIFICATE to backup the certificate.

This behavior is one of the security benefits of TDE. In order to restore the encrypted data to another instance of SQL Server, a user needs to recover the service master key backup file, the Master database master key backup file and the Master database certificate private key, prior to recovering the database backup file.

The database encryption key that is created in the user database, in which TDE has been implemented, is included in the standard database backup. It is stored in the boot record of the database file so that it can be accessed and used to decrypt the user database.

When the service master key and database master key are backed up, it is recommended to store their backup files in a separate location from the database files. This separation will ensure continued protection of the encrypted data in the event that the database backup media is stolen or compromised.

4.2. Implementing TDE to database backup

In this section, we will implement TDE using the HomeLending database. Our TDE implementation, in comparison to cell-level encryption, will be very simple. There are no modifications to the schema required, there are no permissions that need to be granted to database users and roles in order to use TDE, and there are no additional database objects that must be created to perform the encryption and decryption methods

It is a general best practice to backup a database prior to making modifications. However, it is especially important when implementing TDE, in order to ensure that, should the TDE implementation need to be reversed, you can cleanly recover the database in its original form. In addition, by performing a database backup, a new checkpoint will be established in the transaction log. The creation of a checkpoint truncates all inactive items in your transaction log prior to the new checkpoint. This will ensure that your transaction log is free from unencrypted items, prior to the TDE implementation. Listing 1 shows the backup command for the HomeLending database:

```
USE HomeLending;
GO
BACKUP DATABASE HomeLending
  TO DISK = 'D:\HomeLending\Backup\HomeLending.bak'
  WITH NOFORMAT,
  INIT,
  NAME = 'HomeLending-Full Database Backup',
  SKIP,
  NOREWIND,
  NOUNLOAD,
  STATS = 10
GO
```

Listing 1: Backing up the HomeLending database, prior to TDE.

With the backup successfully completed, the implementation of TDE can be proceeded as follow

The Master Database

Our first step is to create a database master key for our Master database,using
the CREATE MASTER KEYmethod, as shown in Listing 2.

```
USE master;
GO
CREATE MASTER KEY
  ENCRYPTION BY PASSWORD = 'MyStr0ngP@ssw0rd2009';
GO
```

Listing 2: Creating the database master key in the Master database.

Notice that, while ENCRYPTED BY PASSWORD is a required argument to the method, our intent, as in Chapter 5, is to instead protect the database master key with the service

ISSN:2277-1581
01 October 2016

master key. This option is automatically available to us, upon creation of the database master key.

A search against the sys.key_encryptions catalog view for the ##MS_DatabaseMasterKey## key, as shown in Listing 3, returns ENCRYPTION BY MASTER KEY, in reference to the service master key.

```
USE master;
GO

SELECT
  b.name,
  a.crypt_type_desc
FROM
  sys.key_encryptions a
  INNER JOIN sys.symmetric_keys b
    ON a.key_id = b.symmetric_key_id
WHERE
  b.name = '##MS_DatabaseMasterKey##';
GO
```

Listing 3: Confirming protection of the database master key by the service master key.

The next step is to create a self-signed certificate that is protected by the database master key of our Masterdatabase. All certificates created within SQL Server, as opposed to being imported, are self-signed. This associates the certificate to the database.
Certificates are created using the
CREATE CERTIFICATE method

```
USE HomeLending;
GO

CREATE CERTIFICATE MyHighCert
  WITH SUBJECT = 'Cert used for sensitive class of high';
GO
```

Listing 4.
Since this certificate is located in the Master database and will be used to protect the database encryption key of our HomeLending database, we will name this certificate MasterCert, as shown in Listing 5.

```
USE master;
GO

CREATE CERTIFICATE MasterCert
  WITH SUBJECT = 'Cert used for TDE';
GO
```

Listing 5: Creating the MasterCert self-signed

As for Listing 4, by omitting the ENCRYPTION BY PASSWORD argument, we are specifying that the certificate is to be protected by the database master key.

At this point in the process you should perform a backup of the certificate with its private key, using the BACKUPCERTIFICATE command shown in Listing 6. In the event that the HomeLending database needs to be restored, this certificate and its private key will be required.

```
USE master;

GO

BACKUP CERTIFICATE MasterCert

   TO FILE = 'D:\HomeLending\Backup\MasterCert.bak'

   WITH PRIVATE KEY (

         FILE = 'D:\HomeLending\Backup\MasterCert.pvk',

                  ENCRYPTION BY
PASSWORD = 'MyB@ckUpP@ssw0rd');

GO
```

Listing 6: Backing up the MasterCert certificate.

Since our MasterCert certificate is protected by the Master database master key, the DECRYPTION BYPASSWORD argument is not included in the WITH PRIVATE KEY argument of this command.

The User Database

Having created the database master key and the MasterCert certificate in the Master database, we are ready to create the database encryption key for the HomeLending database which we will use to perform the cryptographic functions for the physical files of our database.

The database encryption key is created using the CREATE DATABASE ENCRYPTION KEY command. The arguments to this method include:

- WITH ALGORITHM: Specifies the algorithm used, which in turn dictates the strength of the key.
- ENCRYPTION BY: Defines the protection method of the key. The key used in the ENCRYPTION BYargument can be a certificate or an asymmetric key that is located in the Master database.

Listing 7 shows the exact command used for the HomeLending database's database encryption key.

```
USE HomeLending;

GO

 CREATE DATABASE ENCRYPTION KEY
```


doi : 10.17950/ijset/v5s10/1006

ISSN:2277-1581
01 October 2016

```
    WITH ALGORITHM = AES_128

    ENCRYPTION BY SERVER CERTIFICATE MasterCert;

GO
```

Listing 7: Creating the HomeLending database encryption key.

The AES_128 option specifies Advanced Encryption Standard (AES) with a 128 bit key length, and we protect the database encryption key with the MasterCert certificate that was created in the Master database.

The final step in the setup process of TDE is to enable it. This is accomplished by executing the ALTER DATABASE command with the SET ENCRYPTION ON argument.

```
USE HomeLending;

GO

ALTER DATABASE HomeLending

    SET ENCRYPTION ON;

GO
```

Listing 8: Enabling TDE.

At this point, an **encryption scan** occurs, which is the process by which the physical files of the database are scanned and encrypted. Included in this scan process are the database files, TempDB database files and transaction log files.

Transaction log files contain information that is used to maintain data integrity and are used in the restoration process. Within these files are a series of smaller units called **virtual log files** (VLFs). These VLFs contain records that pertain to transactions within the database file. Prior to the implementation of TDE, these VLFs contain unencrypted data. During the encryption scan any pages that have been in the buffer cache and modified, known as**dirty pages**, are written to disk, a new VLF is created and the prior inactive VLFs are truncated. This results in a transaction log that only contains encrypted data.

The duration of the encryption scan will vary depending upon the size of the database files. Once the process has completed, the encryption_state column in the sys.dm_database_encryption_keys dynamic management view will reflect the encryption state of "encrypted", and will show the value of "3" in this column, for our HomeLendingdatabase.

Verifying TDE

Once the implementation of TDE is complete there are a few ways you can verify that these steps indeed succeeded.

Using Dm_Database_Encryption_Keys

Dynamic management views (DMV) are built-in views that provide metadata regarding the settings, health and properties of SQL Server instances and databases. The sys.dm_database_encryption_keys DMV presents information about the database encryption keys used in a given database, as well as the encryption state of the database.

Through the use of a query in which the sys.dm_database_encryption_keys DMV and the sys.databasescatalog view are joined through the database_id column, we are able to determine the success of the TDE implementation, as demonstrated in Listing 9.

```
USE master;

GO
SELECT
   db.name,
   db.is_encrypted,
   dm.encryption_state,
   dm.percent_complete,
   dm.key_algorithm,
   dm.key_length
FROM
   sys.databases db
   LEFT OUTER JOIN sys.dm_database_encryption_keys dm
      ON db.database_id = dm.database_id;
GO
```

Listing9:VerifyingTDE using dm_database_encryption_keys.

A return value of "1" for the is_encrypted column of the sys.databases catalog view indicates that the database has been encrypted through TDE.

III. Results and Tables

Penetration Test Results

After encryption the penetration test was performed to evaluate the vulnerability of the TDE encrypted database backups. The following are results from the test performed.

Vulnerability by risk levels

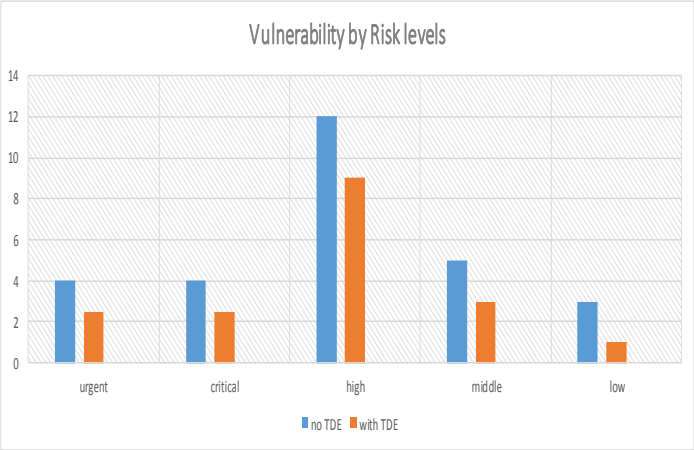

Figure 1 Vulnerability by risk levels

doi : 10.17950/ijset/v5s10/1006

ISSN:2277-1581
01 October 2016

Distribution of findings and their effects

The figure dispays the number of findings and their effects

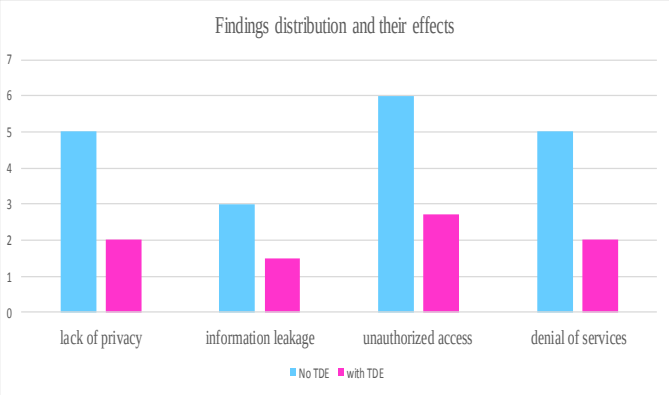

Figure 2 Number of findings and their effects

DISCUSSION

The TDE algorithm for database backup protection was designed and implemented. This showed an improved database backup security. Anwar & Rizayuddin(2011), utilized TDE in the security of database contents where they found that Transparent Data Encryption plays an especially important role in safeguarding data in transit. Based on their study where TDE was developed and implemented, I has been shown that TDE provides highly configurable environment for application development. Therefore this can account for the findings of this study.

The penetration test was performed to assess the vulnerability of transparently data encrypted database backups. As displayed in figure 4 and 5 databases backups without data TDE have higher vulnerability risks than databases with TDE. Christian Kirsch(2009) said that Industry experts have long recommended a "defense in depth" approach by adding layers of security around the data. With the network being regarded as inherently insecure, encrypting the data itself is the best option, often cited as the "last line" of defense. This implies that in terms of database security, encryption secures the actual data within the database and protects backups. That means data remains protected even in the event of a data breach. Modern approaches to database encryption, such as the Transparent Data Encryption (TDE) architectures introduced by Oracle and Microsoft, make it easier for organizations to deploy database encryption because TDE does not require any changes to database applications (Christian Kirsch, 2009). This explains why the study findings showed that backups with TDE showed that they have low vulnerabilities compared to their counterparts without TDE.

The study findings showed that backups with TDE have some vulnerability even though are low. Luc & Yanli (2009), reported that encrypting the data only guarantees data confidentiality, but gives no assurance on data integrity, i.e., on the fact that the data has not been illegally forged or modified (authenticity) or replaced by older versions (freshness). In addition, if the database server is untrusted, (e.g., it may have been tampered by attackers), one should check the query results correctness (results corresponds to the query specification) and completeness (no query result is missing). In his work "Transparent Encryption and Separation of Duties for Enterprise Databases – A Solution for Field Level Privacy in Databases" Ulf Mattson (2009) after using key concept of security, transparent cryptography and proposing solutions on how to transparently store and search encrypted database fields, concluded that people opting to use encryption must include a strategy for protecting sensitive database against attack or misuse of encrypting key data elements.

CONCLUSION

The study had for objectives to design an enhanced transparent data encryption algorithm, apply the designed data protection encryption in backup data and to evaluate the data protection efficacy of the designed enhanced transparent data encryption.

The algorithm was designed and implemented thereafter; the penetration test was done to evaluate the vulnerabilities of encrypted databases backups. The results from this study showed that TDE algorithm for database backup improved database backup security penetration test performed which showed that databases backups without data TDE have higher vulnerability risks than databases with TDE. Therefore protecting backup data with TDE can prevent the exposure of data via backups.

REFERENCES

i. *Iqra Basharat, Farooque Azam & Abdul Wahab Muzaffar, (2012), Database Security and Encryption: A Survey Study, International Journal of Computer Applications (0975 – 888) Volume 47– No.12,*

ii. *Jimmy Thakkar (2015) "Database Security & Encryption: A Survey Study". Journal of Advances in Business Management; Vol. 1, Issue 4, Page: 379-383, DOI: 10.14260/jadbm/2015/47.*

iii. *Khaleel Ahmad; Jayant Shekhar; Nitesh Kumar; K.P. Yadav; (2011), Policy Levels Concerning Database Security; International Journal of Computer Science & Emerging Technologies (E-ISSN: 2044-6004) 368, Volume 2, Issue 3, PP 368-372*

iv. *Luc Bouganim, Yanli Guo. Database encryption. S. Jajodia and H. van Tilborg. Encyclopedia of Cryptography and Security, Springer, pp.1-9, 2009, 978-1-4419-5905-8. <10.1007/978-14419-5906-5 677>. <hal-00623915>*

v. *Christian Kirsch, (2009), The role of encryption in database security, https://www.helpnetsecurity.com/2009/05/13/the-role-of-encryption-in-database-security , accessed on July 2, 2016*

vi. *Ulf T. Mattsson (2004), A Practical Implementation of Transparent Encryption and Separation of Duties In Enterprise Databases Protection against External and Internal Attacks on Databases, https://www.researchgate.net/publication/228280386, accessed on July 12, 2016*

vii. *John Magnabosco, (2010), Transparent Data Encryption, Simple-Talk; A technical journal and community hub from Redgate Redgatehttps://www.simple-talk.com/sql/database-administration/transparent-data-encryption/ accessed on July 7, 2016*